LIGERMOE: EFFICIENT EXPERT PARALLELISM VIA WARP-SPECIALIZED COMMUNICATION-COMPUTE FUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

State-of-the-art large language models rely on Mixture-of-Experts (MoE) layers to scale model capacity while maintaining computational efficiency. At scale, MoE models rely on expert parallelism, whose efficiency is limited by many small expert matrix multiplications and expensive all-to-all communication. To address these bottlenecks, we present persistent, fused MoE kernels for both the forward and backward passes that leverage warp specialization to overlap communication with computation. The kernels use remote direct memory access (RDMA) to access data on remote GPUs without bidirectional sender-receiver coordination, while preallocated symmetric buffers support CUDA Graph execution without requiring a predefined per-expert token capacity. On the Qwen3-30B-A3B MoE model, our forward kernel achieves 49% and 143% higher throughput than the strongest available baselines on $8\times$H200 and $8\times$B300 GPUs, respectively. For the backward pass, our method improves throughput by 31% and 121% on the same systems. In end-to-end BF16 training on $8\times$H200 GPUs, our implementation improves training throughput by 47% compared with Megatron's default MoE implementation.

1 INTRODUCTION

Mixture-of-Experts (MoE) layers are used by many state-of-the-art large language models (Fedus et al., 2022; Dai et al., 2024; Jiang et al., 2024; Yang et al., 2025; Zhu et al., 2024; Kang et al., 2025; Agarwal et al., 2025; Zeng et al., 2026a; Team et al., 2026). They enable models to scale parameter capacity without proportionally increasing per-token computation. Instead of applying all model weights to every token, an MoE layer dynamically routes each token to a small subset of expert networks. This allows MoE models to outperform dense models of similar sizes trained with comparable amounts of compute (Shazeer et al., 2017; Lepikhin et al., 2020; Zhang et al., 2025a).

Large MoE models are typically too large to fit within the memory of a single GPU, necessitating distributed execution across multiple devices. Expert parallelism (EP) addresses this challenge by partitioning expert weights across GPUs and routing token hidden states to the devices hosting their corresponding experts (Rajbhandari et al., 2022; Hwang et al., 2023; Liu et al., 2025). Each MoE layer first dispatches tokens to the corresponding expert GPUs, performs the expert feed-forward-network computations, and then combines the expert outputs on the GPUs that originally owned the tokens. The dispatch and combine phases require expensive all-to-all communication, while dynamic routing produces irregular token distributions and many small expert matrix multiplications (GEMMs) that may underutilize tensor cores in modern GPUs. Furthermore, because of dynamic routing, the number of tokens assigned to each expert is known only after the router has been evaluated. This complicates static execution mechanisms such as CUDA Graphs and XLA, which require kernel launches and memory allocations to remain fixed across iterations (Cai et al., 2025; Yan et al., 2026; Zhang et al., 2026).

Prior works address these challenges through optimized EP communication and communication-computation overlap (Zhao et al., 2025; Aimuyo et al., 2026; Zhang et al., 2025b). However, existing designs either retain separate communication and computation components or assign entire cooperative thread arrays (CTAs) to specialized communication and computation roles, preventing

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

each CTA from contributing to both data movement and tensor-core computation. Extending fusion to the backward pass presents an even harder challenge because it must propagate the input gradients back to the source GPUs while accumulating expert gradients across dynamically routed tokens.

LigerMoE addresses these limitations through warp-specialization within each persistent CTA, allowing every participating CTA to make concurrent progress on communication and expert computation. The resulting contributions are summarized below.

1.1 CONTRIBUTIONS

We introduce LigerMoE, an expert parallel, warp-specialized kernel solution with the following contributions:

- Unlike prior approaches that dedicate separate CTAs to communication and computation, we fuse communication and computation within each persistent CTA using warp-level specialization. This design improves streaming multiprocessor (SM) utilization by enabling both operations to execute concurrently on the same SM.
- We use one-sided RDMA to access routed token data on peer GPUs, instead of bidirectional message-based communication. Symmetric buffers are sized according to the known total number of routed token copies, so the required memory footprint is known before router evaluation. This enables CUDA Graph execution without requiring a predefined per-expert token capacity.
- To the best of our knowledge, this is the first work to implement both the expert computation and communication of MoE forward and backward passes as single persistent CUDA kernels. This design minimizes kernel-launch and intermediate-buffer overheads.

We evaluate LigerMoE against state-of-the-art MoE implementations across a wide range of representative model configurations and input tensor shapes on both Hopper and Blackwell GPUs. For the Qwen3-30B-A3B MoE model, our forward kernel achieves 49% and 143% higher throughput on $8\times$H200 and $8\times$B300 GPUs, respectively, while the backward kernel improves throughput by 31% and 121% on the same systems. Beyond kernel-level benchmarking, we demonstrate end-to-end benefits by integrating our implementation into Megatron-LM, improving BF16 training throughput for Qwen3-30B-A3B model on $8\times$H200 GPUs by approximately 47% while retaining the same loss convergence behavior. Similarly, we observe 7% improvement on training throughput for larger Qwen3.5-122B-A10B model using two $8\times$B200 hosts with an InfiniBand connection. We will make the source code public upon acceptance [1].

2 RELATED WORK

Prior work has improved MoE execution through distributed training systems, load balancing, block-sparse expert computation, communication optimization, and kernel fusion (He et al., 2021; 2022; Gale et al., 2023; Guo et al., 2025; Zhao et al., 2025; Chen et al., 2026; Chang et al., 2024; Zhang et al., 2025b; Sul et al., 2026; Aimuyo et al., 2026; Zheng et al., 2026; Zeng et al., 2026b). We focus on approaches most closely related to expert-parallel communication and fused kernel execution.

Expert-parallel communication and overlap. DeepEP provides specialized dispatch and combine kernels while relying on a separate backend for expert GEMMs (Zhao et al., 2025), whereas MoonEP additionally addresses routing imbalance through dynamic expert replication (Chen et al., 2026). FLUX decomposes communication and GEMM operations into fine-grained tasks to improve overlap (Chang et al., 2024), while DisagMoE overlaps communication and computation through a disaggregated pipeline (Zeng et al., 2026b).

Persistent and fused MoE execution. More closely related to our work, COMET uses dependency-aware scheduling and specialized CTAs to overlap communication with grouped GEMM execution (Zhang et al., 2025b). FlashMoE executes the complete MoE forward pass as

[1] Source code is available in supplementary material and will be open sourced after the review process.

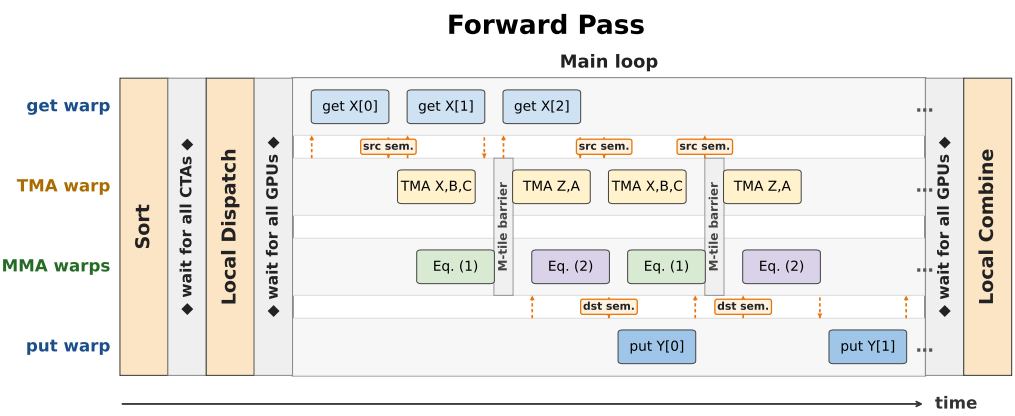

Figure 1: Illustration of the forward pass. All warps perform first radix sort and dispatch to copy the token hidden states into the symmetric memory. This is followed by a main loop that computes the intermediate output Y for all the input token hidden states X - local and remote. The final output O is computed from gathered Y tensor. Here *sem.* refers to semaphore lock for the local buffers.

a single persistent kernel with device-initiated communication (Aimuyo et al., 2026). UniEP extends persistent MegaKernel execution to MoE training by fusing communication-adjacent subgraphs (Zheng et al., 2026).

LigerMoE differs from these approaches in both the granularity and the scope of fusion. Rather than assigning entire CTAs or SM-level workers to communication and computation roles, LigerMoE uses warp-specialization within every persistent CTA, allowing each CTA to advance both workloads concurrently. Furthermore, LigerMoE implements the forward and backward passes as six CUDA kernels where the expert computation and communication is fused into a single persistent kernel for both passes, minimizing kernel launches and intermediate buffer overheads.

3 LigerMoE Kernel

3.1 Forward Pass

We proceed to discuss how LigerMoE kernel computes the forward and backward passes of the MoE layer. Appendix A presents complete set of the forward and backward operations for readers that are interested in the exact formulae. We aim to overlap the dispatch and combine all-scatter steps with the compute. To achieve this, we utilize NVSHMEM[2] library that allows inter GPU communication within the kernel itself. NVSHMEM is Nvidia's device side remote direct memory access (RDMA) library that allows GPUs to access memory from other GPUs. The memory accessible from other GPUs is called *symmetric memory* and needs to be allocated at the same sequence and to the same size across different GPUs referred as processing elements (PE) in NVSHMEM context. Symmetric memory also requires additional coordination that is explained in detail in Appendix B.

Forward pass consists of three kernels: 1) first kernel performs sorting and dispatching tokens to their respective locations in the symmetric memory; 2) the main loop that performs the actual expert computations and data transfer between GPUs; 3) a final combine kernel that reduces the dispatched tokens to the final output. The overall process is illustrated in Fig. 1.

Sort: We first use radix sort to sort all tokens by expert and pad the sorted index array to a multiple of the M-tile size (M_T, 64 or 128 tokens in this work). Sorting is required for tensor cores to process the tokens in tiles instead of individual vectors. We store mapping from token to it's position in the sorted array and the expert offsets in the symmetric memory. Note that each token has multiple experts and appears K times in the index array.

[2]https://github.com/NVIDIA/nvshmem

Local Dispatch: We proceed by copying tokens to their sorted locations in a symmetric memory buffer. For S tokens, K experts per token, and hidden size D we need $S \times K \times D$ storage in the symmetric memory to store all the dispatched tokens. Note that at this point we have not performed any inter GPU data transfer in contrast to regular dispatch step in MoE implementations. As token hidden states are contiguous in memory, the writes can be effectively coalesced resulting in saturation of the HBM bandwidth. After dispatch has completed we wait for all PEs to finish.

Other MoE kernel implementations usually have PEs send their tokens to remote experts which requires complex coordination between the sender and receiver along with memory management (Aimuyo et al., 2026; Zhao et al., 2025; Zhang et al., 2025b). In this work, we instead access the tokens directly from the remote destination without any additional coordination from the origin side using *nvshmem-get* , *nvshmem-put* , or *nvshmem-ptr* APIs. For intra-host GPUs, we use tensor memory accelerator (TMA) to load the remote data and use the *nvshmem-get* API only for the remote hosts (InfiniBand).

Main Loop: As illustrated in Fig. 1, each CTA uses 384 threads that are divided to four roles: i) *nvshmem-get* warp (32 threads); ii) TMA warp (32 threads); iii) matrix-multiply-accumulate (MMA) warps (288 threads); and iv) *nvshmem-put* warp (32 threads).

The *nvshmem-get* warps as discussed before loads the remote data tile block by tile block to local high bandwidth memory (HBM) buffer. The local buffer has multiple stages to avoid blocking the *nvshmem-get* operations due to processing of the tile block. Each tile block consists of $M_T \times D$ elements, and multiple CTAs may collaborate on the same tile block. The communication CTA group size - N_C - transferring data for the same tile block is a hyper-parameter ($N_C = 4$ in this work unless explicitly mentioned otherwise). Each tile block has a semaphore where the *nvshmem-get* warps act as the producers and the TMA warps act as the consumers.

There are several possible ways to iterate over local and remote tile blocks. We choose to traverse the tile blocks in expert order rather than PE order. Specifically, each CTA processes all tile blocks associated with a given expert across all PEs before advancing to the next expert. This ordering significantly improves the L2 cache hit rate for expert weights compared with iterating over tiles in PE order. The improvement arises because a larger number of CTAs concurrently reuse the same expert weights, increasing the likelihood that those weights remain resident in the L2 cache.

Once the tile is ready for processing, we load the tiles from the tile block and associated expert weights to shared memory where they are processed by tensor core MMA instructions issued by the MMA warp(s). In Hopper we use WGMMA instructions that are issued by all the participating warps while in Blackwell series GPUs the corresponding UMMA instructions is issued by a single warp. Appendix C contains more detailed explanation of the Blackwell and Hopper MMA instruction usage. We compute the results in two steps:

$$\mathrm{Z_b} = f(\mathrm{X_b B}_e^T) \odot \mathrm{X_b C}_e^T \quad (1)$$

$$\mathrm{Y_b} = \mathrm{Z_b A}_e^T \quad (2)$$

where e is the tile block's expert and A_e, B_e, and C_e are expert weight matrices; $\odot$ is element wise multiplication; and matrices $\mathrm{Z_b}$, $\mathrm{Y_b}$, and input hidden states $\mathrm{X_b}$ are tile blocks with shape $M_T \times D$. The intermediate result $\mathrm{Z_b}$ is stored in HBM, but the activation is computed using accumulator registers avoiding down casting as well as a round trip to HBM. Each step is performed by a cluster of CTAs with N_G (N_G is chosen based on token count) CTAs collaborating in computing the outputs. The division of work is done by splitting the output tiles between the CTAs.

The output $\mathrm{Y_b}$ writing depends on the location of the receiving GPU. For GPUs connected by NVLink or PCIe bus, the MMA warp issues direct TMA store to the remote memory otherwise the result is written first to the local HBM protected by a semaphore. After all producer CTAs (TMA warps) have finished the Y tiles, the consumer *nvshmem-put* warp issues a put instruction to the InifiniBand network interface card to move the data. For intra-host communication this is unnecessary as the data has already been transferred by the TMA store instructions.

Local Combine: After all the tokens have been received from all the remote experts, we perform the combine step in a separate kernel call that computes the final weighted average of the expert outputs. Reader should note that our combine step does not do any inter GPU communication as the data has already been received at this point. Finally, the symmetric memory matrix X is cached for the backward pass.

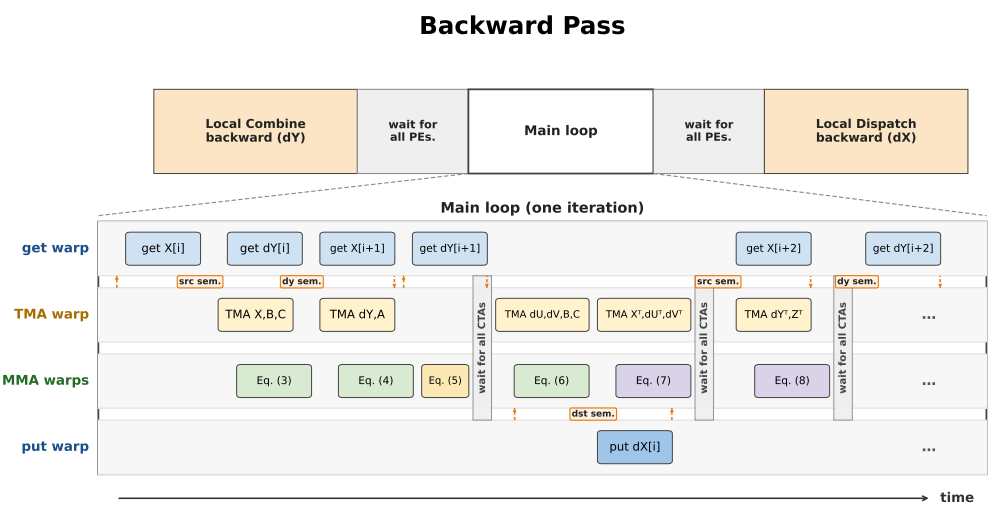

Figure 2: Illustration of the backward pass. All warps begin by computing the expert output and score gradients. This is followed by the main loop that computes the activation gradients followed by the weight gradients. Input activation gradient is computed from the dispatched token gradients.

3.2 Backward Pass

Forward pass operated fundamentally token-by-token that allows processing all the tile blocks independently. This is not the case for the backward pass as the expert weight matrix gradients need a reduction over the tokens (see Appendix A). We approach the problem in similar fashion to the forward pass and load multiple tile blocks across CTAs – referred as tile block batch – that contains $M_T \times M_G$ tokens. Here M_G is the number of CTA groups of size N_G that process the tile blocks collaboratively. The tile block batch semaphores are not released until all CTAs have completed their weight gradient computations (this is required due to the reduction over tokens).

The weight gradient matrices may have tokens from multiple tile block batches which means that we cannot compute the final result in a single iteration. To account for this, we use TMA reduce store instructions to store the weight gradient output tiles instead of regular TMA store. This allows accumulating the gradients over multiple tile block batches with only a modest impact on the throughput.

Fig. 2 shows backward pass flow chart. The **combine backward** kernel computes expert output gradients dY and score gradients dS from the output gradient dO and stores dY in the symmetric memory. Once all PEs have finished this step, we progress to the backward pass main loop. The remote communication part of the backward pass is performed the same way as in the forward pass with an exception that we also need to read tile blocks for dY in addition to the input hidden states X.

The main loop backward matrix computations consist of six parts (see Appendix A for details):

$$\mathrm{U}_b = \mathrm{X}_b\mathrm{B}_e^T;\ \mathrm{V}_b = \mathrm{X}_b\mathrm{C}_e^T;\ \mathrm{Z}_b = f(\mathrm{U}_b) \odot \mathrm{V}_b \tag{3}$$

$$\mathrm{dZ}_b = \mathrm{dY}_b\mathrm{A}_e \tag{4}$$

$$\mathrm{dU}_b = \mathrm{dZ}_b \odot f'(\mathrm{U}_b) \odot \mathrm{V}_b;\ \mathrm{dV}_b = \mathrm{dZ}_b \odot f(\mathrm{U}_b) \tag{5}$$

$$\mathrm{dX}_b = \mathrm{dU}_b\mathrm{B}_e + \mathrm{dV}_b\mathrm{C}_e \tag{6}$$

$$\mathrm{dB}_b = \mathrm{dU}_b^T\mathrm{X}_b;\ \mathrm{dC}_b = \mathrm{dV}_b^T\mathrm{X}_b \tag{7}$$

$$\mathrm{dA}_b = \mathrm{dY}_b^T\mathrm{Z}_b. \tag{8}$$

The Eq. (3) is recomputation of partial forward pass since we cannot cache these values. The order of operations also matters. The *nvshmem-put* warp can start writing the results after Eq. (6) has completed, and *nvshmem-get* warp can start reading next X_b after Eq. (7) has finished.

The dimensions of Eq. (3) to Eq. (6) compared with Eq. (7) and Eq. (8) can be widely different. For this reason we use two different CTA grids for the computation; $M_G \times N_G$ is used to process the token tile block based computations while we use a second grid $M_F \times N_F$ for the weight matrix gradients. The grid shapes are treated as hyper-parameters and need to satisfy $M_G \times N_G = M_F \times N_F = M_C \times N_C$ where $M_C \times N_C$ is the communication warp CTA grid. The values are found using a grid search and heuristics described in detail in Appendix D.

In the backward pass, the MMA warps write all output tiles to local HBM rather than directly to remote intra-host memory. We found this approach to be faster because, in almost all cases, the put warp data transfer is overlapped with the weight-gradient computation. In the forward pass, no analogous computation is available to hide this transfer, leaving the final put warp data transfer fully exposed in the kernel execution time.

Local Dispatch: After all the tokens have been processed and copied to the symmetric memory, each GPU can compute the input gradients by reducing over all the dispatched token gradients. We opted not to pipeline the final dispatch with the remote token transport as it is significantly faster than the main loop and the increased coordination is larger overhead than any gains from starting it with partial tokens.

4 Experimental Setup

We compare LigerMoE against existing MoE implementations in both standalone kernel benchmarks and end-to-end training experiments. Specifically, we report results for Comet (Zhang et al., 2025b), Transformer-Engine, DeepEP with a Transformer-Engine backend (Zhao et al., 2025), and FlashMoE (Aimuyo et al., 2026). All frameworks were compiled and evaluated in the same software and hardware environment on Hopper- and Blackwell-generation GPUs. Complete environment details are provided in Appendix E. We validate the numerical accuracy of LigerMoE against a Torch implementation using the same numerical precision. LigerMoE achieves slightly lower relative errors than the PyTorch reference; detailed results are provided in Appendix F.

5 Results and Discussion

5.1 Kernel Experiments

Figures 3 and 4 compare the estimated (model) floating-point operations per second (FLOPS) of different MoE implementations on H200 and B300 GPUs, respectively. DeepEP and Comet are omitted from the single-GPU experiments because they require at least two GPUs. Likewise, Comet and FlashMoE are excluded from the B300 experiments because they do not support the Blackwell architecture. DeepEP is excluded from the inter-host experiments[3]. Backward pass comparison is restricted to Transformer-Engine and DeepEP as the other methods do not have dedicated backward pass implementations. Complete results of all methods is provided in Appendices G and H.

5.1.1 Forward Pass

We find that our proposed MoE implementation is at least $\sim 50\%$ faster than FlashMoE which is closest to our forward pass implementation in design, as it is also overlapping the communication and computation using a single kernel. We observe similar result with Comet, but the gap between our method and Comet is smaller - we observed throughput improvement ranging from 10% to 45%. Both Comet and FlashMoE use CTA specialization for communication i.e one CTA either does communication or computation. This limitation leads to either idle tensor cores (CTA is transferring data) or underutilizing the bandwidth (CTA is not participating in data transfer) as Nvidia GPUs usually avoid scheduling two persistent kernels in the same SM. Our approach of using warp specialization avoids this by having each SM contribute to both.

DeepEP's MoE implementation uses a separate backend for matrix multiplications (Transformer-Engine in our experiments) and is responsible only for data transfer. We observed that it did not

[3]DeepEP supports InfiniBand but does not support the CPU assisted IBGDA backend at the time of writing, which was available in our hosts. Hence, we could only use it in intra-host experiments.

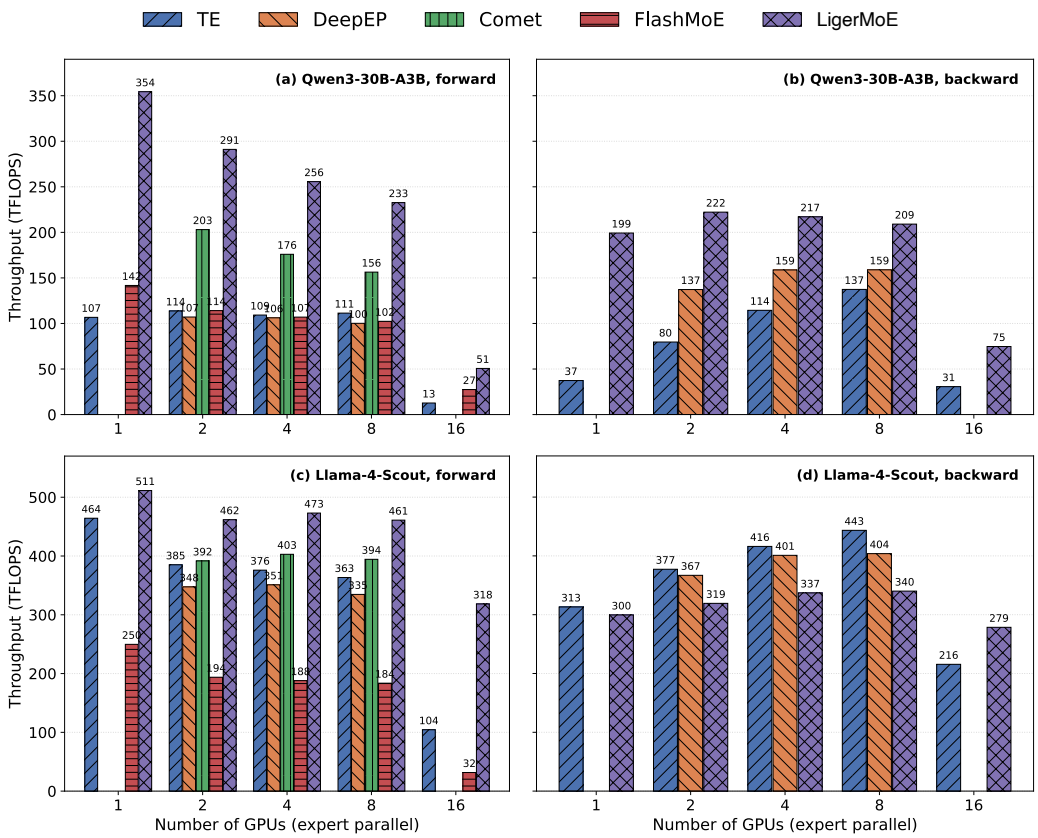

Figure 3: FLOPS for various MoE implementations with different H200 GPU counts in expert parallel setting, with the number of tokens set to 8192. Top: Qwen3-30B-A3B ($D = 2048$, $I = 768$). Bottom: Llama4 scout ($D = 5120$, $I = 8192$). Left: forward pass. Right: backward pass.

outperform standard NCCL-based communication routines for dispatch and combine. We therefore do not discuss it further in the context of the forward pass.

Our kernel outperforms Transformer-Engine's grouped GEMM kernel in all evaluated cases on both H200 and B300 GPUs (see Appendix G). The performance gains are generally larger on Blackwell than on Hopper. On Hopper, our kernel dedicates additional registers to the communication warps, reducing the register capacity available for WGMMA accumulators. This can limit tensor-core throughput for very large GEMMs in Hopper.

5.1.2 Backward Pass

We classify LLMs as either communication- or compute-intensive based on the ratio of the intermediate dimension (I) to the hidden dimension (D). The computational cost scales approximately as $\sim I \times D$, whereas the communication volume depends primarily on the hidden-state size. Consequently, models with a large intermediate dimension relative to the hidden dimension (e.g., Scout and Mixtral) tend to be more compute-bound, while models with a smaller intermediate dimension (e.g., Qwen) tend to be more communication-bound.

Unlike in the forward pass, a clear performance pattern emerges in the backward pass. In communication-bound cases, our method achieves up to $\sim 2\times$ higher throughput than Transformer-Engine. In compute-bound cases, however, this trend reverses, with Transformer-Engine achieving up to $\sim 30\%$ higher throughput. For inter-host configurations (16 GPUs), which are inherently more communication-bound, LigerMoE achieves higher throughput than all other backends across the evaluated models, with Llama4 Scout being the only exception.

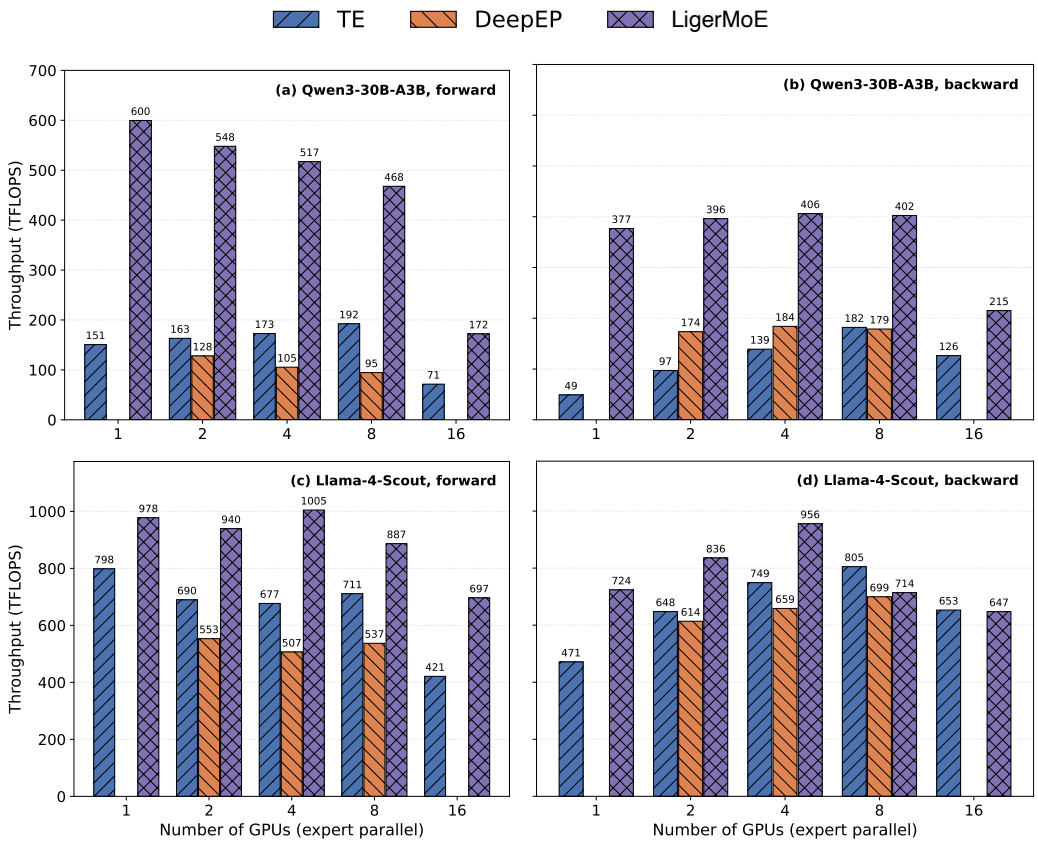

Figure 4: FLOPS for various MoE implementations with different B300 GPU counts. See Fig. 3 caption for additional details.

The backward-pass limitation in compute-bound cases (Scout and Mixtral) arises from the same register-pressure constraints associated with Hopper's WGMMA instructions that affect the forward pass. Because the backward pass is intrinsically more complex than the forward pass and requires additional intermediate state and computations, these register-pressure effects become more pronounced. This explains the larger relative performance gap observed in compute-bound workloads during the backward pass.

5.2 End-to-End Training

Fig. 5 shows cross-entropy training loss and per-step wall time across 300 training steps for different backend implementations. Table 1 shows a complete statistical breakdown of step times and loss improvement. The smaller Qwen3 model is trained with single host expert parallelism using Hopper while the larger Qwen3.5 model is using expert parallelism over two hosts.

For the intra-host case, Liger outperforms other backend implementation in training step time with a 47% speedup, compared with TE's 41% speed-up in relation to the Megatron baseline on Hopper devices. For multi-host Blackwell setup, we find that the TE and default Megatron are more closely matched, but Liger is 7% faster in end-to-end training than the Megatron. The training losses of all three methods are essentially the same.

6 Conclusion

We present persistent, fused expert-parallel MoE kernels for both the forward and backward passes. To the best of our knowledge, this is the first backward-pass implementation to fuse communica-

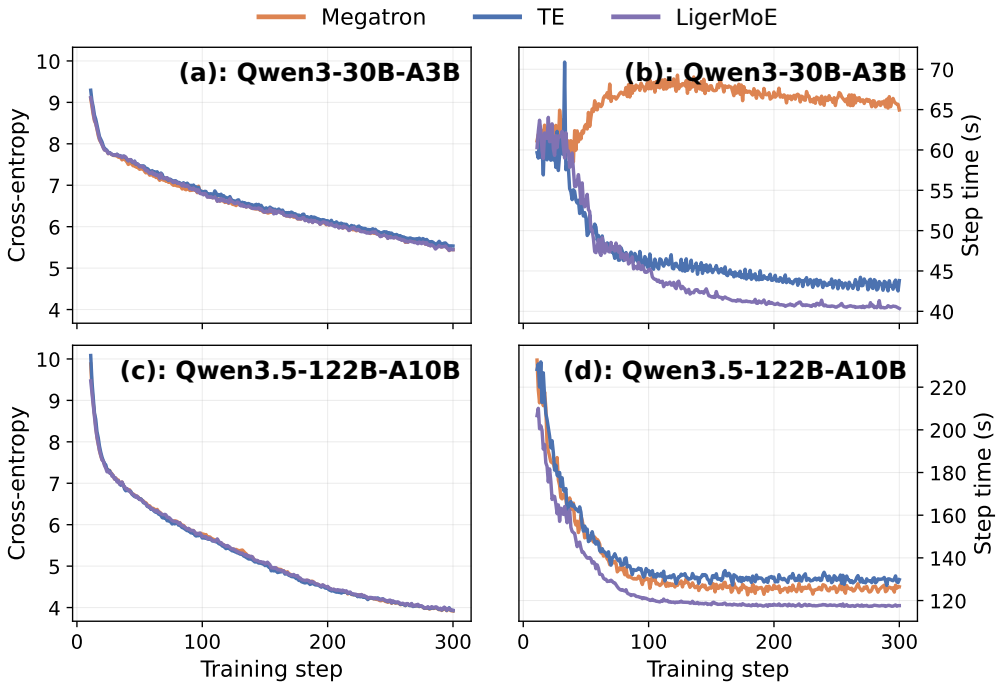

Figure 5: Pre-training comparison of Megatron, Transformer Engine (TE) and Liger with OpenWebText data (Gokaslan et al., 2019). Sequence length is set to 8192 with 300 training steps. (a) and (b): Qwen3-30B-A3B using 8× H200 GPUs, $EP = 8$, and global batch size of 256. (c) and (d): Qwen3.5-122B-A10B using 16× B200 GPUs, $EP = 16$ and global batch size of 512.

Table 1: Metric comparison for pre-training using 8 H200 GPUs for Qwen3-30B-A3B and 16 B200 GPUs for Qwen3.5-122B-A10B. Step-time statistics are over steps 10–300.

Model	Backend	Step time (ms) mean	min	max	std	Speedup (×)	Final loss
Qwen3-30B-A3B	Megatron	66,154	58,514	69,274	2,086	1.000	5.446845
	TE	46,731	42,521	70,892	4,904	1.416	5.533610
	LigerMoE	44,936	40,356	64,042	6,382	1.472	5.444666
Qwen3.5-122B-A10B	Megatron	135,351	122,657	232,697	20,035	1.000	3.916712
	TE	139,211	127,067	231,865	19,893	0.972	3.927942
	LigerMoE	126,489	117,234	210,093	18,067	1.070	3.934512

tion and computation within a single CTA. Our design combines warp specialization with RDMA to overlap communication and computation while allowing all CTAs to participate in both tasks, in contrast to prior approaches that dedicate separate CTAs to communication. Furthermore, our implementation integrates naturally with CUDA Graphs without imposing arbitrary token-capacity constraints.

Across a broad range of workloads, our kernels consistently outperform existing MoE implementations. For the Qwen3-30B-A3B MoE model, our forward kernel achieves 49% and 143% higher throughput on 8×H200 and 8×B300 GPUs, respectively, while the backward kernel improves throughput by 31% and 121% on the same systems[4]. We further integrate our implementation into the Megatron framework for end-to-end training, improving BF16 training throughput by 47% relative to the default Megatron backend for Qwen3-30B-A3B using eight H200 GPUs.

[4]Compared with Comet on Hopper and Transformer-Engine on Blackwell GPUs.

AI USE STATEMENT

In this work, we used generative AI tools to assist with grammatical editing, documentation search, and source code development. We did not use generative AI tools to generate experimental results; all results reported in this manuscript were obtained from experiments conducted on real hardware as described in the paper.

All AI-assisted content, including source code and editorial revisions, was reviewed and validated by the authors. We take full responsibility for the final content of this work, including all text, claims, results, and artifacts produced with the assistance of generative AI tools.

REPRODUCIBILITY STATEMENT

We only use open source datasets and models in this work. The source code is available in the supplementary material and will be open sourced in GitHub later. We also list our exact software and hardware environments in Appendix E so the compute environment can be accurately reproduced.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Osayamen Aimuyo, Byungsoo Oh, and Rachee Singh. Flashmoe: Fast distributed moe in a single kernel. *Advances in Neural Information Processing Systems*, 38:100676–100699, 2026.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. Flux: Fast software-based communication overlap on gpus through kernel fusion, 2024.

Yutian Chen, Cong Li, Yucheng Wang, and Ming Wei. Moonep: A perfectly balanced expert parallelism library via dynamic redundant experts. `https://github.com/MoonshotAI/MoonEP`, 2026.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5:288–304, 2023.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Wentao Guo, Mayank Mishra, Xinle Cheng, Ion Stoica, and Tri Dao. Sonicmoe: Accelerating moe with io and tile-aware optimizations. *arXiv preprint arXiv:2512.14080*, 2025.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021.

Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 120–134, 2022.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Hao Kang, Zichun Yu, and Chenyan Xiong. Flame-moe: A transparent end-to-end research platform for mixture-of-experts language models. *arXiv preprint arXiv:2505.20225*, 2025.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Dennis Liu, Zijie Yan, Xin Yao, Tong Liu, Vijay Korthikanti, Evan Wu, Shiqing Fan, Gao Deng, Hongxiao Bai, Jianbin Chang, et al. Moe parallel folding: Heterogeneous parallelism mappings for efficient large-scale moe model training with megatron core. *arXiv preprint arXiv:2504.14960*, 2025.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Stuart H. Sul, Nash Brown, Henry Wildermuth, William Lin, and Federico Cassano. Mixture-of-kittens: MoE megakernel for NVL72s. `https://github.com/cursor/mixture-of-kittens`, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Zijie Yan, Hongxiao Bai, Xin Yao, Dennis Liu, Tong Liu, Hongbin Liu, Pingtian Li, Evan Wu, Shiqing Fan, Li Tao, et al. Scalable training of mixture-of-experts models with megatron core. *arXiv preprint arXiv:2603.07685*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026a.

Zhichen Zeng, Chi-Chih Chang, Jiayi Wang, Zezhou Wang, Ningxin Zheng, Zheng Zhong, Cesar A Stuardo, Dongyang Wang, Mohamed S Abdelfattah, Haibin Lin, et al. Disagmoe: Computation-communication overlapped moe training via disaggregated af-pipe parallelism. *arXiv preprint arXiv:2605.11005*, 2026b.

Danyang Zhang, Junhao Song, Ziqian Bi, Xinyuan Song, Yingfang Yuan, Tianyang Wang, Joe Yeong, and Junfeng Hao. Mixture of experts in large language models. *arXiv preprint arXiv:2507.11181*, 2025a.

Qi Zhang, Jidong Zhai, and Weimin Zheng. A survey on accelerated technologies for mixture-of-experts model training systems. *Tsinghua Science and Technology*, 31(3):1411–1439, 2026.

Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, et al. Comet: Fine-grained computation-communication overlapping for mixture-of-experts. *Proceedings of Machine Learning and Systems*, 7, 2025b.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. Deepep: an efficient expert-parallel communication library. `https://github.com/deepseek-ai/DeepEP`, 2025.

Size Zheng, Xuegui Zheng, Li-wen Chang, and Jidong Zhai. Uniep: Unified expert-parallel moe megakernel for llm training. *arXiv preprint arXiv:2604.19241*, 2026.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 15913–15923, 2024.

A MIXTURE-OF-EXPERT LAYERS

Mixture-of-Expert models use a variation of the multi-layer-perceptron (MLP) that has E individual expert MLPs instead of a single MLP as in dense transformer models. More formally, we have routing weight matrix $\mathrm{W} \in \mathbb{R}^{E\times D}$; and weight matrices $\{\mathrm{A}_e \in \mathbb{R}^{D\times I}; \mathrm{B}_e, \mathrm{C}_e \in \mathbb{R}^{I\times D} | e \in [1, E]\}$, where D is the hidden (input) dimension and I is the intermediate dimension.

We start by computing scores for each input token and extracting top-K scores along with the expert indices:

$$\begin{aligned} \mathrm{S}_{ij} &= \mathrm{X}_{il}\mathrm{W}_{jl} && (9)\\ \mathcal{I}_{ik} &= \mathrm{TopK}(\mathrm{S}_{i:}, k), && (10) \end{aligned}$$

where $\mathrm{X} \in \mathbb{R}^{S\times D}$ is the input hidden states, S is the number of tokens, i, j, k are regular indices, l refers to Einstein summation index, and $\mathrm{TopK}(\mathrm{S}_{i:}, k)$ operator selects k:th largest element from $\mathrm{S}_{i:}$. We use the top-K experts to compute MLP outputs for each input token, and finally average them using the scores S to get the final output O:

$$\begin{aligned} \mathrm{U}_{ijk} &= \mathrm{X}_{il}\mathrm{B}_{\mathcal{I}_{ij},kl} && (11)\\ \mathrm{V}_{ijk} &= \mathrm{X}_{il}\mathrm{C}_{\mathcal{I}_{ij},kl} && (12)\\ \mathrm{Z}_{ijk} &= f(\mathrm{U}_{ijk})\mathrm{V}_{ijk} && (13)\\ \mathrm{Y}_{ijk} &= \mathrm{Z}_{ijl}\mathrm{A}_{\mathcal{I}_{ij},kl} && (14)\\ \mathrm{O}_{ij} &= \mathrm{S}_{il}\mathrm{Y}_{ilj}, && (15) \end{aligned}$$

where $f(x) = x\sigma(x)$ is SiLU function. Gradients for the backward pass of the weight matrices can be obtained by applying chain-rule from Eq. (11) to Eq. (15). For this we will adopt following notation:

$$\mathrm{dO}_{ij} \equiv \frac{\partial \mathcal{L}}{\partial \mathrm{O}_{ij}}, \tag{16}$$

where $\mathcal{L}$ is the loss function. With this notation, the gradient formulae become:

$$\begin{aligned} \mathrm{dS}_{ij} &= \mathrm{dO}_{il}\mathrm{Y}_{ilj} && (17)\\ \mathrm{dY}_{ijk} &= \mathrm{dO}_{ik}\mathrm{S}_{ij} && (18)\\ \mathrm{dZ}_{ijk} &= \mathrm{dY}_{ijl}\mathrm{A}_{\mathcal{I}_{ij},lk} && (19)\\ \mathrm{dU}_{ijk} &= \mathrm{dZ}_{ijk}f'(\mathrm{U}_{ijk})\mathrm{V}_{ijk} && (20)\\ \mathrm{dV}_{ijk} &= \mathrm{dZ}_{ijk}f(\mathrm{U}_{ijk}) && (21)\\ \mathrm{dX}_{ij} &= \mathrm{dU}_{ilp}\mathrm{B}_{\mathcal{I}_{il},pj} + \mathrm{dV}_{ilp}\mathrm{C}_{\mathcal{I}_{il},pj}, && (22) \end{aligned}$$

where indices p and l refer to Einstein summation. For the remaining gradients dA_{ijk}, dB_{ijk}, dC_{ijk} we define index set $\mathcal{I}_i^{-1}$ which consists of all index tuples that map to the expert i. More formally lets define following set:

$$\mathcal{I}_i^{-1} = \{(p, q) | \text{ for all } (p, q) \text{ where } \mathcal{I}_{uv} = i\}. \tag{23}$$

This yields following weight gradients:

$$\mathrm{dA}_{ijk} = \sum_{(p,q)\in\mathcal{I}_i^{-1}} \mathrm{dY}_{pqj}\mathrm{Z}_{pqk} \tag{24}$$

$$\mathrm{dB}_{ijk} = \sum_{(p,q)\in\mathcal{I}_i^{-1}} \mathrm{dU}_{pqj}\mathrm{X}_{pk} \tag{25}$$

$$\mathrm{dC}_{ijk} = \sum_{(p,q)\in\mathcal{I}_i^{-1}} \mathrm{dV}_{pqj}\mathrm{X}_{pk}. \tag{26}$$

B Symmetric Memory Management

The symmetric memory used to store dispatched tokens is allocated during initialization and persists throughout training. If each token is dispatched to K experts, the maximum number of tokens processed during training is S, and the hidden dimension is D, the required symmetric memory capacity is $K \times S \times D$ elements.

Unlike conventional tensor memory, symmetric memory is impractical to allocate and deallocate dynamically because of its higher allocation cost and the coordination required across GPUs. Instead, we manage symmetric memory using a simple memory pool consisting of two stacks: one containing available memory chunks and another tracking chunks currently in use. During the forward pass, the kernel requests two memory chunks to store the dispatched tokens and the corresponding expert outputs. If no free chunk is available, a new symmetric memory allocation is created and added to the pool.

During the backward pass, the kernel either retrieves the stored intermediate tensors from the forward pass or recomputes them when gradient checkpointing is enabled. Once the corresponding symmetric memory is no longer needed, the allocated chunks are returned to the pool of available memory. Consequently, the calling code must distinguish between training and inference: during training, the memory is released only after the associated backward pass completes, whereas during inference it can be returned to the pool immediately after the forward pass.

The primary drawback of this approach is that it reserves memory for the worst-case token count on every GPU, even though the actual number of dispatched tokens may differ across devices. However, this design eliminates additional synchronization overhead, remains fully compatible with graph execution frameworks, and keeps the memory management logic simple.

C MMA Warp Structure

Hopper and Blackwell GPUs expose different tensor-core programming models. Hopper uses warpgroup matrix multiply-accumulate (WGMMA) instructions, which accumulate intermediate results in registers. High-performance Hopper kernels typically dedicate 256 threads (two warp groups) to issuing WGMMA instructions and performing the epilogue[5]. Because WGMMA accumulates directly into registers owned by the issuing warp group, the producer and consumer warps are tightly coupled, making it difficult to pipeline computation across independent output tiles.

Blackwell replaces WGMMA with universal matrix multiply-accumulate (UMMA) instructions. Unlike WGMMA, UMMA stores accumulators in dedicated tensor memory (TMEM) rather than registers. TMEM can be accessed by other warps through `tcgen05` load and store instructions, decoupling matrix multiplication from subsequent processing. This organization enables efficient pipelining across output tiles, as accumulator values can remain in TMEM without requiring register transfers between warps. In our implementation, a single warp issues the UMMA instructions, while the remaining MMA warps (256 threads) perform the epilogue, including the SiLU activation and stores to HBM.

[5]This structure is used by high-performance kernel libraries such as CUTLASS and Quack.

Table 2: Versions of all packages used in this study. For off-release dependencies or dependencies that do not have official releases, we used the GitHub commit hash as the version.

Package	Version
CUDA Runtime	12.9
cuDNN	9.17.0
Cutlass	4.4.1
NVSHMEM	3.6.5-0
NCCL	2.28.9
torch	2.9
Transformer-Engine	2.10.0
Megatron-LM	0.15
FlashMoE	9042973
Comet	19831ca
DeepEP	567632d

D HYPER-PARAMETER TUNING

We implemented the algorithm discussed in Sections 3.1 and 3.2 using the Cutlass library. Unlike Triton and other runtime-compiled frameworks, Cutlass is a CUDA library whose source code must be compiled before the actual input matrix shapes are known. Each MMA operation is characterized by its tile sizes (M-tile, N-tile, and K-tile) as well as the number of stages. Once three of these four parameters are fixed, the register file size and available shared memory determine the fourth. In addition, the forward and backward passes must share the same sequence tile size, creating a coupling between the two passes.

We created a set of hyperparameter templates for both kernels across a large collection of input shapes. Each input tensor configuration is characterized by the number of dispatched tokens ($K \times S$), the hidden dimension, and the intermediate dimension. For every valid template and input configuration, we measured the forward and backward compute times to identify the template with the lowest execution time. The resulting pairs of input tensor shapes and optimal template hyper-parameters were then stored in a static array.

During runtime, the actual input shapes may not exactly match those used during tuning. To address this, we computed the Euclidean distance between the logarithms of the runtime input shape parameters and those of the tuned configurations. After identifying the closest tuned input shape, we selected the corresponding kernel hyper-parameters for execution.

The computation grid spanning over the token range (M_G) is chosen inside the kernel after the GPUs actual token count is known. The choice is based on a simple heuristic that minimizes idle CTA time. The benefit of this approach is that we can adjust the grids independently across different ranks in cases where there is token imbalance across the workers - a common situation in MoEs.

E RUNTIME ENVIRONMENT

We used the CUDA 12.9 runtime and compilation environment for all evaluated packages. All experiments were conducted using NVIDIA H200 (Hopper), B200 or B300 (Blackwell) GPUs with BF16 numerical precision. GPUs within each host were interconnected via NVLink, with the available bandwidth determined by the GPU architecture, while inter-host communication used InfiniBand links. H200 hosts were provisioned with 8 GPUs and four $50, \mathrm{GB/s}$ links; B200 and B300 hosts were provisioned with eight GPUs and $50, \mathrm{GB/s}$ or $100, \mathrm{GB/s}$ links, respectively. InfiniBand communication used CPU assisted IBGDA backend on all systems. Table 2 summarizes the software packages used in our experiments together with their corresponding versions.

We introduced $250, \mathrm{ms}$ cooldown intervals between consecutive kernel executions to minimize run-to-run variability caused by GPU frequency throttling. This ensures that all implementations are evaluated under comparable clock-frequency conditions and that the reported measurements reflect differences in kernel execution rather than transient changes in GPU operating frequency. In the end-to-end training cases we did not apply any cooldown periods.

F NUMERICAL ACCURACY

We measured the numerical accuracy of LigerMoE with ground truth derived from torch implementation using higher FP32 numerical precision. We also share the accuracy metrics for the same implementation using lower BF16 precision to give readers more context. Results for eight H200 are shown in Table 3 and for eight B300 in Table 4.

We can see that LigerMoE has approximately $\sim 25\%$ lower relative error than the reference torch implementation (using BF16). This originates from performing the activation computations directly in registers without down casting the accumulator values to BF16 from FP32 when storing it in HBM.

Table 3: Routed SwiGLU MoE accuracy against Torch FP32, using identical BF16-representable inputs, weights, routes and upstream gradients. Relative mean and max are $\mathrm{mean}/\max|z - z_{32}|/\max(\max|z_{32}|, \epsilon)$, aggregated over all PEs. EP=8; seed=42; $\epsilon = 10^{-12}$. Tensor definitions are given in Appendix A. Computations were done using $8\times$H200 GPUs.

Model	Tokens/PE	Tensor	LigerMoE BF16		Torch BF16	
			Rel. mean	Rel. max	Rel. mean	Rel. max
Qwen3-30B-A3B	1024	O	2.828e-04	4.562e-03	4.578e-04	7.336e-03
Qwen3-30B-A3B	1024	dX	4.305e-04	6.961e-03	5.137e-04	6.671e-03
Qwen3-30B-A3B	1024	dA	2.659e-04	6.651e-03	2.928e-04	5.407e-03
Qwen3-30B-A3B	1024	dB	1.936e-04	4.600e-03	2.206e-04	5.218e-03
Qwen3-30B-A3B	1024	dC	1.622e-04	5.015e-03	2.402e-04	8.089e-03
Qwen3-30B-A3B	1024	dS	4.320e-04	3.015e-03	6.956e-04	4.546e-03
Qwen3-30B-A3B	2048	O	2.823e-04	4.387e-03	4.570e-04	6.596e-03
Qwen3-30B-A3B	2048	dX	3.883e-04	5.622e-03	4.632e-04	6.194e-03
Qwen3-30B-A3B	2048	dA	3.110e-04	5.425e-03	3.073e-04	5.563e-03
Qwen3-30B-A3B	2048	dB	3.565e-04	5.909e-03	3.646e-04	7.490e-03
Qwen3-30B-A3B	2048	dC	3.178e-04	6.133e-03	3.987e-04	9.278e-03
Qwen3-30B-A3B	2048	dS	4.768e-04	3.634e-03	7.640e-04	5.267e-03
Qwen3-235B-A22B	1024	O	2.904e-04	4.606e-03	4.702e-04	5.528e-03
Qwen3-235B-A22B	1024	dX	3.945e-04	4.841e-03	4.708e-04	5.207e-03
Qwen3-235B-A22B	1024	dA	2.231e-04	6.682e-03	2.457e-04	6.648e-03
Qwen3-235B-A22B	1024	dB	2.587e-04	6.139e-03	2.947e-04	6.284e-03
Qwen3-235B-A22B	1024	dC	1.442e-04	5.402e-03	2.135e-04	6.468e-03
Qwen3-235B-A22B	1024	dS	4.977e-04	3.422e-03	8.016e-04	5.141e-03
Qwen3-235B-A22B	2048	O	3.043e-04	4.550e-03	4.932e-04	7.217e-03
Qwen3-235B-A22B	2048	dX	3.937e-04	6.594e-03	4.697e-04	6.594e-03
Qwen3-235B-A22B	2048	dA	3.577e-04	6.816e-03	3.857e-04	7.036e-03
Qwen3-235B-A22B	2048	dB	2.404e-04	4.962e-03	2.684e-04	4.639e-03
Qwen3-235B-A22B	2048	dC	1.939e-04	5.023e-03	2.780e-04	5.345e-03
Qwen3-235B-A22B	2048	dS	5.194e-04	4.052e-03	8.341e-04	5.276e-03
Llama-4-Scout	1024	O	3.090e-04	3.049e-03	5.294e-04	4.452e-03
Llama-4-Scout	1024	dX	4.745e-04	4.095e-03	6.032e-04	5.658e-03
Llama-4-Scout	1024	dA	3.721e-04	4.958e-03	4.226e-04	6.037e-03
Llama-4-Scout	1024	dB	3.338e-04	4.403e-03	3.947e-04	5.433e-03
Llama-4-Scout	1024	dC	2.547e-04	3.767e-03	4.356e-04	5.547e-03
Llama-4-Scout	1024	dS	5.536e-04	3.308e-03	8.832e-04	4.465e-03
Llama-4-Scout	2048	O	3.024e-04	3.144e-03	5.181e-04	4.433e-03
Llama-4-Scout	2048	dX	4.516e-04	4.090e-03	5.739e-04	5.959e-03
Llama-4-Scout	2048	dA	4.233e-04	6.679e-03	4.662e-04	4.860e-03
Llama-4-Scout	2048	dB	3.996e-04	5.512e-03	4.588e-04	4.872e-03
Llama-4-Scout	2048	dC	2.846e-04	4.610e-03	4.606e-04	4.960e-03
Llama-4-Scout	2048	dS	4.963e-04	3.178e-03	7.882e-04	4.111e-03

Table 4: Routed SwiGLU MoE accuracy against Torch FP32, using identical BF16-representable inputs, weights, routes and upstream gradients. Relative mean and max are $\mathrm{mean}/\max|z - z_{32}|/\max(\max|z_{32}|, \epsilon)$, aggregated over all PEs. EP=8; seed=42; $\epsilon = 10^{-12}$. Tensor definitions are given in Appendix A. Computations were done using 8×B300 GPUs.

Model	Tokens/PE	Tensor	LigerMoE BF16		Torch BF16	
			Rel. mean	Rel. max	Rel. mean	Rel. max
Qwen3-30B-A3B	1024	O	2.518e-04	4.538e-03	4.079e-04	6.436e-03
Qwen3-30B-A3B	1024	dX	3.689e-04	6.281e-03	4.396e-04	5.936e-03
Qwen3-30B-A3B	1024	dA	2.832e-04	6.985e-03	2.992e-04	5.931e-03
Qwen3-30B-A3B	1024	dB	2.013e-04	6.651e-03	2.198e-04	6.577e-03
Qwen3-30B-A3B	1024	dC	2.211e-04	5.793e-03	3.274e-04	8.173e-03
Qwen3-30B-A3B	1024	dS	4.697e-04	3.368e-03	7.551e-04	4.943e-03
Qwen3-30B-A3B	2048	O	2.817e-04	4.741e-03	4.561e-04	6.472e-03
Qwen3-30B-A3B	2048	dX	4.570e-04	5.375e-03	5.455e-04	7.019e-03
Qwen3-30B-A3B	2048	dA	4.035e-04	8.320e-03	3.938e-04	6.133e-03
Qwen3-30B-A3B	2048	dB	3.988e-04	8.113e-03	4.035e-04	6.487e-03
Qwen3-30B-A3B	2048	dC	2.578e-04	6.017e-03	3.772e-04	7.169e-03
Qwen3-30B-A3B	2048	dS	4.624e-04	3.436e-03	7.456e-04	4.397e-03
Qwen3-235B-A22B	1024	O	3.220e-04	5.496e-03	5.218e-04	7.410e-03
Qwen3-235B-A22B	1024	dX	4.692e-04	5.416e-03	5.598e-04	7.304e-03
Qwen3-235B-A22B	1024	dA	2.590e-04	6.047e-03	2.871e-04	6.778e-03
Qwen3-235B-A22B	1024	dB	2.522e-04	6.046e-03	2.890e-04	7.161e-03
Qwen3-235B-A22B	1024	dC	1.970e-04	5.973e-03	2.946e-04	8.004e-03
Qwen3-235B-A22B	1024	dS	4.638e-04	3.236e-03	7.428e-04	4.445e-03
Qwen3-235B-A22B	2048	O	2.477e-04	4.547e-03	4.012e-04	6.048e-03
Qwen3-235B-A22B	2048	dX	4.030e-04	5.701e-03	4.807e-04	5.913e-03
Qwen3-235B-A22B	2048	dA	3.385e-04	6.578e-03	3.677e-04	6.130e-03
Qwen3-235B-A22B	2048	dB	2.799e-04	6.197e-03	3.148e-04	5.934e-03
Qwen3-235B-A22B	2048	dC	2.189e-04	5.699e-03	3.177e-04	6.987e-03
Qwen3-235B-A22B	2048	dS	4.832e-04	3.822e-03	7.776e-04	4.613e-03
Llama-4-Scout	1024	O	3.220e-04	3.309e-03	5.517e-04	4.717e-03
Llama-4-Scout	1024	dX	4.652e-04	4.269e-03	5.914e-04	7.550e-03
Llama-4-Scout	1024	dA	3.470e-04	6.615e-03	3.941e-04	6.415e-03
Llama-4-Scout	1024	dB	3.676e-04	5.816e-03	4.349e-04	5.816e-03
Llama-4-Scout	1024	dC	2.523e-04	3.674e-03	4.316e-04	5.654e-03
Llama-4-Scout	1024	dS	5.974e-04	4.051e-03	9.509e-04	4.839e-03
Llama-4-Scout	2048	O	3.199e-04	3.381e-03	5.483e-04	4.924e-03
Llama-4-Scout	2048	dX	4.495e-04	3.906e-03	5.714e-04	6.423e-03
Llama-4-Scout	2048	dA	3.970e-04	5.699e-03	4.403e-04	4.974e-03
Llama-4-Scout	2048	dB	4.047e-04	6.493e-03	4.676e-04	5.734e-03
Llama-4-Scout	2048	dC	2.838e-04	4.611e-03	4.651e-04	4.993e-03
Llama-4-Scout	2048	dS	5.816e-04	3.713e-03	9.464e-04	5.001e-03

G FORWARD PASS RUNTIMES

Tables 5 and 6 show MoE forward pass FLOPS and execution time for all the MoE implementations using various model hyper-parameters as well as GPU counts and architecture. Comet and FlashMoE did not compile for Blackwell series GPUs so we excluded those from the B300 GPU results. FLOPS are computed by dividing the number of floating point operations required by the MoE matrix multiplications with the kernel computation time.

Tables 7 and 8 show MoE forward pass FLOPS and execution time using H200 GPUs for all the MoE implementations for generic input shapes with varying hidden dimensions, intermediate dimensions, expert counts and active expert counts.

Table 5: Named-model MoE forward throughput at $T = 8192$ tokens/rank using H200 GPUs. Each cell is TFLOPS / latency, where latency is per-microbatch in ms; bold marks the fastest backend per row.

Model	GPU	TE	DeepEP	Comet	FlashMoE	LigerMoE
Llama-4-Scout	1	464 / 4.44	–	–	250 / 8.26	**511 / 4.03**
	2	385 / 5.36	348 / 5.93	392 / 5.26	194 / 10.64	**462 / 4.47**
	4	376 / 5.49	351 / 5.87	403 / 5.12	188 / 10.97	**473 / 4.36**
	8	363 / 5.67	335 / 6.16	394 / 5.23	184 / 11.23	**461 / 4.47**
	16	104 / 19.74	–	–	32 / 65.17	**318 / 6.48**
Mixtral-8x7B	1	584 / 9.89	–	–	256 / 22.53	**679 / 8.50**
	2	518 / 11.15	496 / 11.65	454 / 12.72	233 / 24.78	**601 / 9.60**
	4	493 / 11.72	470 / 12.30	435 / 13.28	224 / 25.74	**617 / 9.36**
	8	466 / 12.39	459 / 12.57	414 / 13.94	218 / 26.50	**553 / 10.44**
	16	–	–	–	–	–
Mixtral-8x22B	1	636 / 15.55	–	–	263 / 37.62	**694 / 14.25**
	2	561 / 17.65	537 / 18.44	480 / 20.64	236 / 41.89	**611 / 16.19**
	4	518 / 19.10	496 / 19.96	448 / 22.07	219 / 45.27	**585 / 16.93**
	8	466 / 21.22	457 / 21.67	409 / 24.17	199 / 49.85	**514 / 19.26**
	16	–	–	–	–	–
Qwen3-30B-A3B	1	107 / 5.80	–	–	142 / 4.37	**354 / 1.75**
	2	114 / 5.44	107 / 5.78	203 / 3.05	114 / 5.43	**291 / 2.13**
	4	109 / 5.67	106 / 5.83	176 / 3.52	107 / 5.79	**256 / 2.42**
	8	111 / 5.56	100 / 6.18	156 / 3.96	102 / 6.05	**233 / 2.66**
	16	13 / 48.73	–	–	27 / 22.56	**51 / 12.22**
Qwen3-235B-A22B	1	247 / 10.01	–	–	192 / 12.92	**505 / 4.90**
	2	236 / 10.49	233 / 10.63	363 / 6.81	162 / 15.28	**435 / 5.69**
	4	215 / 11.50	219 / 11.31	319 / 7.76	156 / 15.86	**406 / 6.10**
	8	194 / 12.77	193 / 12.82	269 / 9.19	140 / 17.64	**372 / 6.66**
	16	25 / 100.96	–	–	25 / 99.85	**90 / 27.61**
Qwen3.5-35B-A3B	1	47 / 8.78	–	–	111 / 3.71	**248 / 1.66**
	2	51 / 8.13	58 / 7.10	152 / 2.72	74 / 5.55	**196 / 2.10**
	4	70 / 5.87	68 / 6.11	132 / 3.12	70 / 5.87	**155 / 2.67**
	8	74 / 5.56	68 / 6.11	113 / 3.65	66 / 6.26	**144 / 2.87**
	16	8 / 48.54	–	–	21 / 19.70	**34 / 12.15**
Qwen3.5-122B-A10B	1	128 / 9.64	–	–	157 / 7.89	**356 / 3.48**
	2	139 / 8.94	136 / 9.09	239 / 5.17	109 / 11.32	**321 / 3.85**
	4	147 / 8.45	146 / 8.50	201 / 6.16	106 / 11.63	**286 / 4.33**
	8	152 / 8.17	142 / 8.69	173 / 7.17	104 / 11.93	**272 / 4.55**
	16	17 / 72.72	–	–	25 / 48.79	**67 / 18.36**

H Backward Pass Runtimes

Tables 9 and 10 show MoE backward pass FLOPS and execution time for selected MoE implementations using multiple model hyper-parameters and GPU counts and architecture. Comet and FlashMoE did not compile for Blackwell series GPUs so we excluded those from the B300 GPU results.

Tables 11 and 12 show MoE forward pass FLOPS and execution time using H200 GPUs for all the MoE implementations for generic input shapes with varying hidden dimensions, intermediate dimensions, expert counts and active expert counts.

I Runtime with Sequence Length

Figs. 6 and 7 show result of different MoE implementations with different number of tokens, using H200 and B300 GPUs respectively. We can observe that all implementations throughput increases with larger token counts. This can be explained by more complete utilization of the SMs. LigerMoE pipelines the token transport and compute in the main loop with a chunk size ~ 4000 tokens (see Fig. 1). The chunk size is the result of N_C CTA groups copying the tokens from remote destination to the local compute buffers. Higher token count deepen the pipeline stages allowing for a better overlap of the communication and compute.

Table 6: Named-model MoE forward throughput at $T = 8192$ tokens/rank using B300 GPUs. Each cell is TFLOPS / latency, where latency is per-microbatch in ms; bold marks the fastest backend per row.

Model	GPU	TE	DeepEP	LigerMoE
Llama-4-Scout	1	798 / 2.58	–	**978 / 2.11**
	2	690 / 2.99	553 / 3.73	**940 / 2.19**
	4	677 / 3.05	507 / 4.07	**1005 / 2.05**
	8	711 / 2.90	537 / 3.84	**887 / 2.33**
	16	421 / 4.90	–	**697 / 2.96**
Mixtral-8x7B	1	1097 / 5.26	–	**1313 / 4.40**
	2	979 / 5.90	835 / 6.92	**1283 / 4.50**
	4	977 / 5.91	800 / 7.21	**1324 / 4.36**
	8	948 / 6.09	761 / 7.58	**1176 / 4.91**
	16	–	–	–
Mixtral-8x22B	1	1241 / 7.98	–	**1246 / 7.94**
	2	1118 / 8.85	995 / 9.95	**1329 / 7.45**
	4	1061 / 9.33	891 / 11.11	**1244 / 7.96**
	8	982 / 10.08	810 / 12.22	**1174 / 8.43**
	16	–	–	–
Qwen3-30B-A3B	1	151 / 4.11	–	**600 / 1.03**
	2	163 / 3.79	128 / 4.84	**548 / 1.13**
	4	173 / 3.58	105 / 5.87	**517 / 1.20**
	8	192 / 3.22	95 / 6.55	**468 / 1.32**
	16	71 / 8.66	–	**172 / 3.60**
Qwen3-235B-A22B	1	396 / 6.25	–	**858 / 2.89**
	2	368 / 6.72	313 / 7.90	**838 / 2.95**
	4	359 / 6.89	270 / 9.17	**822 / 3.01**
	8	342 / 7.24	228 / 10.87	**762 / 3.25**
	16	115 / 21.46	–	**265 / 9.32**
Qwen3.5-35B-A3B	1	66 / 6.28	–	**367 / 1.12**
	2	81 / 5.11	62 / 6.71	**335 / 1.23**
	4	93 / 4.44	62 / 6.63	**297 / 1.39**
	8	119 / 3.46	62 / 6.70	**274 / 1.51**
	16	47 / 8.69	–	**113 / 3.67**
Qwen3.5-122B-A10B	1	175 / 7.07	–	**602 / 2.06**
	2	217 / 5.71	167 / 7.39	**604 / 2.05**
	4	226 / 5.47	161 / 7.67	**565 / 2.19**
	8	262 / 4.72	151 / 8.18	**551 / 2.25**
	16	94 / 13.18	–	**226 / 5.47**

Table 7: MoE forward throughput sweep over shape $D{=}I$, experts E and top-k K at $T = 8192$ on 8 H200 GPUs (expert parallel). Each cell is TFLOPS / latency(ms); bold marks the fastest backend per row.

D/I	E	K	TE	DeepEP	Comet	FlashMoE	LigerMoE
2048/2048	8	1	79 / 2.61	84 / 2.45	116 / 1.77	133 / 1.55	**197 / 1.05**
		2	130 / 3.18	130 / 3.18	188 / 2.19	158 / 2.62	**287 / 1.44**
		4	178 / 4.62	166 / 4.96	225 / 3.67	181 / 4.55	**343 / 2.40**
	16	1	80 / 2.58	78 / 2.66	118 / 1.75	128 / 1.61	**186 / 1.11**
		2	127 / 3.24	131 / 3.15	185 / 2.23	158 / 2.61	**281 / 1.47**
		4	183 / 4.50	170 / 4.86	233 / 3.54	174 / 4.73	**336 / 2.46**
		8	226 / 7.31	202 / 8.17	263 / 6.27	177 / 9.35	**401 / 4.11**
	32	1	78 / 2.65	77 / 2.67	113 / 1.83	113 / 1.82	**181 / 1.14**
		2	123 / 3.36	120 / 3.44	189 / 2.19	146 / 2.83	**267 / 1.55**
		4	184 / 4.48	169 / 4.89	241 / 3.42	165 / 4.99	**352 / 2.34**
		8	229 / 7.20	209 / 7.89	268 / 6.15	171 / 9.65	**384 / 4.29**
	64	1	79 / 2.62	80 / 2.59	105 / 1.96	91 / 2.27	**163 / 1.27**
		2	124 / 3.32	128 / 3.23	182 / 2.27	114 / 3.63	**252 / 1.63**
		4	181 / 4.56	173 / 4.77	238 / 3.47	146 / 5.67	**346 / 2.38**
		8	230 / 7.16	215 / 7.67	269 / 6.14	163 / 10.11	**439 / 3.76**
	128	1	73 / 2.84	71 / 2.90	97 / 2.13	49 / 4.24	**148 / 1.39**
		2	121 / 3.41	117 / 3.51	159 / 2.59	95 / 4.35	**228 / 1.81**
		4	179 / 4.60	162 / 5.09	231 / 3.57	122 / 6.74	**333 / 2.48**
		8	233 / 7.08	213 / 7.73	273 / 6.04	151 / 10.93	**425 / 3.88**
4096/4096	8	1	221 / 3.73	216 / 3.81	275 / 3.00	180 / 4.58	**415 / 1.99**
		2	283 / 5.82	268 / 6.15	347 / 4.75	195 / 8.44	**448 / 3.69**
		4	313 / 10.54	285 / 11.57	345 / 9.56	185 / 17.87	**478 / 6.90**
	16	1	221 / 3.74	206 / 4.00	269 / 3.07	178 / 4.64	**373 / 2.21**
		2	285 / 5.79	270 / 6.10	356 / 4.64	198 / 8.34	**454 / 3.63**
		4	308 / 10.73	291 / 11.34	340 / 9.69	181 / 18.19	**471 / 7.01**
		8	298 / 22.13	287 / 23.02	314 / 21.04	165 / 39.91	**418 / 15.79**
	32	1	216 / 3.82	204 / 4.04	259 / 3.18	140 / 5.88	**335 / 2.46**
		2	286 / 5.77	269 / 6.12	361 / 4.57	174 / 9.46	**440 / 3.75**
		4	319 / 10.35	305 / 10.81	362 / 9.13	186 / 17.74	**481 / 6.86**
		8	281 / 23.46	275 / 23.96	298 / 22.17	148 / 44.46	**415 / 15.91**
	64	1	212 / 3.90	203 / 4.06	235 / 3.51	116 / 7.13	**293 / 2.81**
		2	283 / 5.83	277 / 5.95	359 / 4.59	143 / 11.51	**402 / 4.10**
		4	335 / 9.84	320 / 10.31	378 / 8.73	181 / 18.22	**482 / 6.85**
		8	304 / 21.72	303 / 21.77	329 / 20.07	160 / 41.29	**449 / 14.69**
	128	1	204 / 4.04	201 / 4.10	194 / 4.26	59 / 14.06	**243 / 3.40**
		2	276 / 5.97	268 / 6.15	316 / 5.23	118 / 14.02	**353 / 4.68**
		4	337 / 9.80	323 / 10.20	393 / 8.40	150 / 21.98	**459 / 7.19**
		8	328 / 20.09	332 / 19.86	365 / 18.07	171 / 38.68	**491 / 13.44**

Table 8: MoE forward throughput sweep over shape $D{=}I$, experts E and top-k K at $T = 8192$ on 8 B300 GPUs (expert parallel). Each cell is TFLOPS / latency(ms); bold marks the fastest backend per row.

D/I	E	K	TE	DeepEP	LigerMoE
2048/2048	8	1	162 / 1.27	107 / 1.93	**441 / 0.47**
		2	237 / 1.74	139 / 2.98	**616 / 0.67**
		4	341 / 2.42	165 / 5.01	**695 / 1.19**
	16	1	152 / 1.35	105 / 1.97	**391 / 0.53**
		2	234 / 1.77	140 / 2.94	**622 / 0.66**
		4	338 / 2.44	176 / 4.69	**683 / 1.21**
		8	403 / 4.10	208 / 7.94	**844 / 1.96**
	32	1	149 / 1.38	104 / 1.98	**396 / 0.52**
		2	234 / 1.76	143 / 2.89	**605 / 0.68**
		4	338 / 2.44	179 / 4.60	**749 / 1.10**
		8	419 / 3.94	221 / 7.47	**826 / 2.00**
	64	1	145 / 1.42	99 / 2.07	**355 / 0.58**
		2	226 / 1.83	137 / 3.01	**519 / 0.79**
		4	322 / 2.56	176 / 4.68	**708 / 1.16**
		8	422 / 3.91	226 / 7.31	**826 / 2.00**
	128	1	131 / 1.58	91 / 2.27	**270 / 0.76**
		2	212 / 1.95	131 / 3.14	**440 / 0.94**
		4	323 / 2.56	176 / 4.69	**618 / 1.33**
		8	412 / 4.01	223 / 7.40	**773 / 2.14**
4096/4096	8	1	414 / 1.99	298 / 2.77	**901 / 0.92**
		2	532 / 3.10	361 / 4.57	**873 / 1.89**
		4	582 / 5.67	365 / 9.04	**997 / 3.31**
	16	1	411 / 2.01	292 / 2.83	**736 / 1.12**
		2	547 / 3.01	372 / 4.43	**942 / 1.75**
		4	584 / 5.65	384 / 8.58	**958 / 3.44**
		8	574 / 11.49	385 / 17.13	**956 / 6.90**
	32	1	403 / 2.05	288 / 2.86	**726 / 1.14**
		2	530 / 3.11	372 / 4.44	**853 / 1.93**
		4	610 / 5.41	410 / 8.05	**1002 / 3.29**
		8	547 / 12.06	386 / 17.09	**715 / 9.23**
	64	1	376 / 2.20	274 / 3.02	**601 / 1.37**
		2	533 / 3.10	377 / 4.37	**773 / 2.13**
		4	633 / 5.21	434 / 7.60	**905 / 3.65**
		8	579 / 11.41	420 / 15.69	**781 / 8.45**
	128	1	360 / 2.29	266 / 3.10	**453 / 1.82**
		2	503 / 3.28	356 / 4.63	**685 / 2.41**
		4	636 / 5.19	436 / 7.57	**889 / 3.71**
		8	617 / 10.69	461 / 14.32	**1009 / 6.54**

Table 9: Named-model MoE backward throughput at $T = 8192$ tokens/rank using H200 GPUs. Each cell is TFLOPS / latency, where latency is per-microbatch in ms; bold marks the fastest backend per row.

Model	GPU	TE	DeepEP	LigerMoE
Llama-4-Scout	1	**313 / 17.56**	–	300 / 18.35
	2	**377 / 14.57**	367 / 14.98	319 / 17.22
	4	**416 / 13.21**	401 / 13.71	337 / 16.31
	8	**443 / 12.40**	404 / 13.61	340 / 16.16
	16	216 / 25.50	–	**279 / 19.74**
Mixtral-8x7B	1	**466 / 33.02**	–	464 / 33.18
	2	**469 / 32.79**	460 / 33.50	449 / 34.26
	4	**470 / 32.76**	453 / 34.00	447 / 34.43
	8	**455 / 33.86**	441 / 34.94	407 / 37.79
	16	–	–	–
Mixtral-8x22B	1	**524 / 50.32**	–	467 / 56.49
	2	**513 / 51.49**	503 / 52.42	405 / 65.19
	4	**490 / 53.81**	480 / 55.00	383 / 68.87
	8	**453 / 58.28**	446 / 59.19	399 / 66.21
	16	–	–	–
Qwen3-30B-A3B	1	37 / 44.15	–	**199 / 8.28**
	2	80 / 20.71	137 / 12.02	**222 / 7.42**
	4	114 / 14.41	159 / 10.39	**217 / 7.60**
	8	137 / 12.01	159 / 10.38	**209 / 7.89**
	16	31 / 53.85	–	**75 / 22.08**
Qwen3-235B-A22B	1	141 / 46.72	–	**291 / 22.64**
	2	201 / 32.82	275 / 24.02	**302 / 21.85**
	4	221 / 29.79	286 / 23.03	**331 / 19.96**
	8	220 / 30.01	270 / 24.46	**300 / 22.03**
	16	55 / 119.88	–	**118 / 55.83**
Qwen3.5-35B-A3B	1	13 / 85.49	–	**148 / 7.42**
	2	29 / 37.49	67 / 16.46	**164 / 6.72**
	4	49 / 22.52	93 / 11.80	**169 / 6.50**
	8	81 / 13.55	112 / 9.81	**173 / 6.34**
	16	21 / 53.40	–	**53 / 20.74**
Qwen3.5-122B-A10B	1	37 / 89.10	–	**223 / 14.78**
	2	84 / 39.37	159 / 20.71	**212 / 15.56**
	4	124 / 26.61	192 / 17.14	**251 / 13.17**
	8	166 / 19.89	207 / 15.94	**268 / 12.30**
	16	40 / 82.61	–	**97 / 33.98**

Table 10: Named-model MoE backward throughput at $T = 8192$ tokens/rank using B300 GPUs. Each cell is TFLOPS / latency, where latency is per-microbatch in ms; bold marks the fastest backend per row.

Model	GPU	TE	DeepEP	LigerMoE
Llama-4-Scout	1	471 / 11.67	–	**724 / 7.60**
	2	648 / 8.49	614 / 8.96	**836 / 6.58**
	4	749 / 7.34	659 / 8.35	**956 / 5.75**
	8	**805 / 6.83**	699 / 7.86	714 / 7.70
	16	**653 / 8.42**	–	647 / 8.49
Mixtral-8x7B	1	814 / 18.91	–	**967 / 15.92**
	2	847 / 18.16	802 / 19.19	**1025 / 15.01**
	4	863 / 17.85	795 / 19.36	**1061 / 14.51**
	8	838 / 18.36	771 / 19.97	**956 / 16.10**
	16	–	–	–
Mixtral-8x22B	1	978 / 26.99	–	**1099 / 24.02**
	2	983 / 26.84	965 / 27.36	**1101 / 23.96**
	4	947 / 27.86	904 / 29.17	**1046 / 25.22**
	8	887 / 29.77	838 / 31.48	**1018 / 25.92**
	16	–	–	–
Qwen3-30B-A3B	1	49 / 33.72	–	**377 / 4.38**
	2	97 / 16.98	174 / 9.50	**396 / 4.16**
	4	139 / 11.85	184 / 8.96	**406 / 4.06**
	8	182 / 9.06	179 / 9.22	**402 / 4.10**
	16	126 / 13.05	–	**215 / 7.67**
Qwen3-235B-A22B	1	157 / 41.95	–	**557 / 11.85**
	2	249 / 26.51	395 / 16.71	**692 / 9.54**
	4	308 / 21.42	398 / 16.56	**704 / 9.37**
	8	330 / 19.97	365 / 18.06	**646 / 10.21**
	16	188 / 35.01	–	**326 / 20.25**
Qwen3.5-35B-A3B	1	17 / 63.37	–	**229 / 4.79**
	2	39 / 28.01	87 / 12.65	**264 / 4.17**
	4	62 / 17.69	100 / 11.00	**288 / 3.81**
	8	94 / 11.68	114 / 9.62	**290 / 3.79**
	16	81 / 13.63	–	**152 / 7.23**
Qwen3.5-122B-A10B	1	49 / 67.22	–	**355 / 9.29**
	2	101 / 32.62	208 / 15.87	**468 / 7.04**
	4	155 / 21.21	247 / 13.38	**519 / 6.36**
	8	215 / 15.36	259 / 12.76	**534 / 6.18**
	16	154 / 21.37	–	**291 / 11.34**

Table 11: MoE backward throughput sweep over shape $D{=}I$, experts E and top-k K at $T = 8192$ on 8 H200 GPUs (expert parallel). Each cell is TFLOPS / latency(ms); bold marks the fastest backend per row.

D/I	E	K	TE	DeepEP	LigerMoE
2048/2048	8	1	188 / 2.93	144 / 3.81	**227 / 2.42**
		2	236 / 4.67	202 / 5.44	**290 / 3.79**
		4	255 / 8.61	232 / 9.46	**320 / 6.87**
	16	1	182 / 3.02	147 / 3.74	**220 / 2.50**
		2	222 / 4.95	193 / 5.69	**278 / 3.96**
		4	252 / 8.71	241 / 9.13	**316 / 6.97**
		8	254 / 17.35	256 / 17.16	**323 / 13.60**
	32	1	170 / 3.23	141 / 3.89	**199 / 2.76**
		2	210 / 5.23	197 / 5.58	**261 / 4.21**
		4	244 / 9.02	236 / 9.31	**313 / 7.03**
		8	254 / 17.34	262 / 16.81	**328 / 13.42**
	64	1	141 / 3.91	132 / 4.16	**172 / 3.20**
		2	196 / 5.62	187 / 5.89	**230 / 4.79**
		4	229 / 9.60	236 / 9.31	**289 / 7.60**
		8	250 / 17.61	268 / 16.43	**323 / 13.61**
	128	1	97 / 5.69	104 / 5.28	**147 / 3.74**
		2	156 / 7.06	165 / 6.67	**200 / 5.50**
		4	207 / 10.61	217 / 10.12	**253 / 8.69**
		8	238 / 18.51	261 / 16.82	**314 / 14.00**
4096/4096	8	1	355 / 6.19	299 / 7.35	**359 / 6.13**
		2	360 / 12.22	340 / 12.93	**364 / 12.09**
		4	341 / 25.79	344 / 25.59	**383 / 22.94**
	16	1	339 / 6.49	298 / 7.37	**348 / 6.32**
		2	**361 / 12.17**	351 / 12.52	360 / 12.21
		4	332 / 26.50	341 / 25.83	**363 / 24.22**
		8	304 / 57.85	318 / 55.29	**325 / 54.08**
	32	1	**316 / 6.96**	289 / 7.60	290 / 7.57
		2	**354 / 12.41**	345 / 12.75	352 / 12.49
		4	343 / 25.65	355 / 24.77	**376 / 23.39**
		8	284 / 62.05	300 / 58.71	**317 / 55.55**
	64	1	**281 / 7.84**	264 / 8.34	242 / 9.08
		2	334 / 13.15	**340 / 12.95**	328 / 13.40
		4	352 / 25.00	**369 / 23.81**	353 / 24.92
		8	305 / 57.64	328 / 53.64	**343 / 51.32**
	128	1	222 / 9.92	**232 / 9.47**	207 / 10.63
		2	296 / 14.84	**307 / 14.30**	261 / 16.88
		4	339 / 25.97	**363 / 24.21**	341 / 25.80
		8	323 / 54.41	358 / 49.15	**364 / 48.27**

Table 12: MoE backward throughput sweep over shape $D{=}I$, experts E and top-k K at $T = 8192$ on 8 B300 GPUs (expert parallel). Each cell is TFLOPS / latency(ms); bold marks the fastest backend per row.

D/I	E	K	TE	DeepEP	LigerMoE
2048/2048	8	1	335 / 1.64	192 / 2.86	**456 / 1.21**
		2	374 / 2.94	248 / 4.44	**547 / 2.01**
		4	399 / 5.51	288 / 7.64	**607 / 3.62**
	16	1	309 / 1.78	191 / 2.87	**397 / 1.38**
		2	361 / 3.04	245 / 4.49	**546 / 2.02**
		4	385 / 5.71	296 / 7.42	**607 / 3.62**
		8	396 / 11.10	331 / 13.27	**674 / 6.52**
	32	1	258 / 2.13	185 / 2.98	**383 / 1.44**
		2	331 / 3.32	244 / 4.51	**504 / 2.18**
		4	380 / 5.79	295 / 7.45	**594 / 3.70**
		8	389 / 11.32	343 / 12.83	**630 / 6.98**
	64	1	197 / 2.79	166 / 3.31	**327 / 1.68**
		2	270 / 4.07	232 / 4.75	**442 / 2.49**
		4	352 / 6.25	291 / 7.55	**560 / 3.92**
		8	388 / 11.34	345 / 12.76	**623 / 7.06**
	128	1	114 / 4.80	138 / 3.99	**244 / 2.26**
		2	204 / 5.39	210 / 5.23	**364 / 3.02**
		4	284 / 7.75	271 / 8.10	**511 / 4.30**
		8	365 / 12.03	340 / 12.94	**594 / 7.40**
4096/4096	8	1	601 / 3.66	469 / 4.69	**805 / 2.73**
		2	612 / 7.19	531 / 8.28	**751 / 5.85**
		4	580 / 15.17	515 / 17.07	**861 / 10.22**
	16	1	582 / 3.78	462 / 4.75	**621 / 3.54**
		2	608 / 7.23	540 / 8.14	**816 / 5.39**
		4	564 / 15.60	522 / 16.84	**813 / 10.82**
		8	524 / 33.54	505 / 34.86	**816 / 21.56**
	32	1	529 / 4.16	444 / 4.95	**603 / 3.65**
		2	592 / 7.43	526 / 8.37	**712 / 6.18**
		4	578 / 15.22	554 / 15.89	**842 / 10.45**
		8	493 / 35.66	486 / 36.22	**610 / 28.83**
	64	1	435 / 5.05	392 / 5.61	**482 / 4.56**
		2	546 / 8.06	517 / 8.51	**631 / 6.97**
		4	589 / 14.93	574 / 15.34	**728 / 12.08**
		8	519 / 33.87	525 / 33.52	**644 / 27.31**
	128	1	315 / 6.98	347 / 6.34	**352 / 6.25**
		2	447 / 9.85	470 / 9.35	**536 / 8.21**
		4	551 / 15.97	561 / 15.69	**690 / 12.74**
		8	543 / 32.41	568 / 30.96	**798 / 22.06**

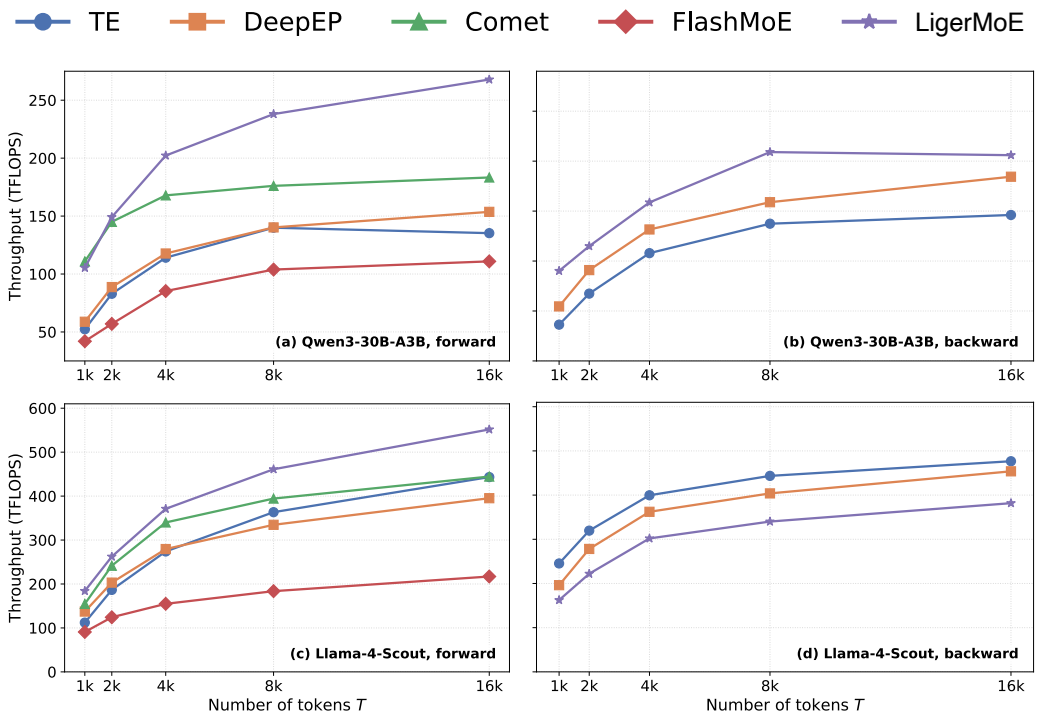

Figure 6: MoE throughput with different sequence lengths using 8 H200 GPUs (expert parallel). Each line is a backend. Top: Qwen3-30B-A3B (D=2048, I=768, E=128, K=8). Bottom: Llama4 Scout (D=5120, I=8192, E=16, K=1).

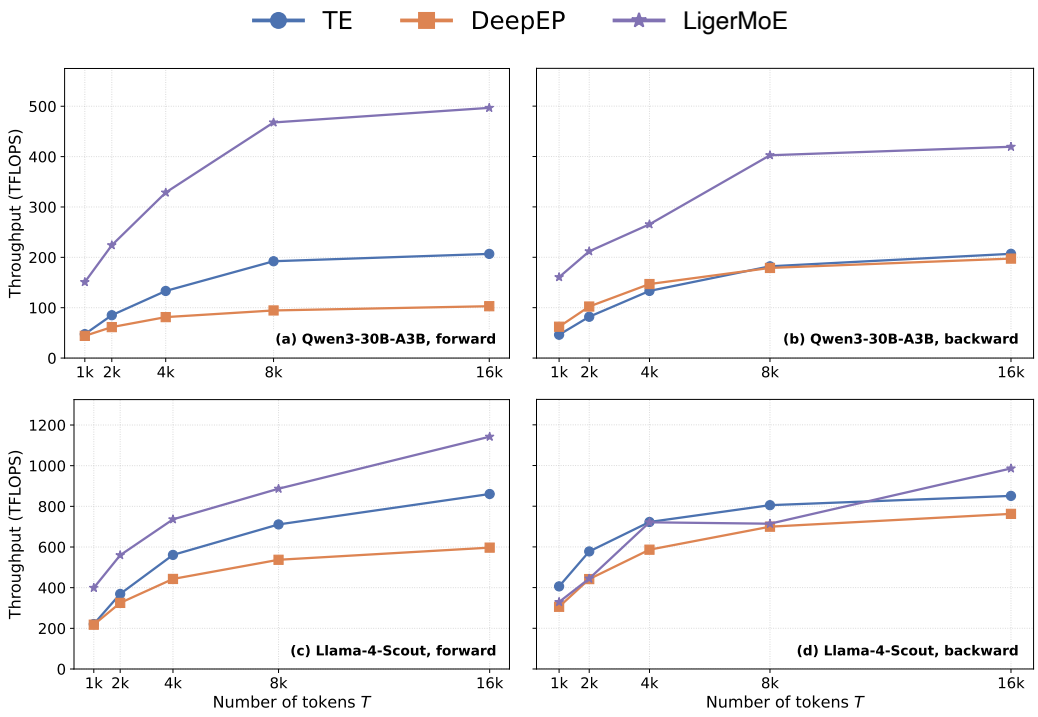

Figure 7: MoE throughput with different sequence lengths using 8 B300 GPUs (expert parallel). Each line is a backend. Top: Qwen3-30B-A3B (D=2048, I=768, E=128, K=8). Bottom: Llama4 Scout (D=5120, I=8192, E=16, K=1).